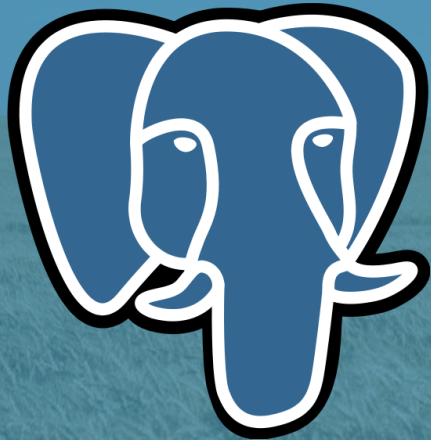


What we (already) know about PostgreSQL 11



Daniel Westermann

Delivery Manager

Open Infrastructure Technology Leader

+41 79 927 24 46

daniel.westermann[at]dbi-services.com



Who we are



Experts At Your Service

- > Over 60 specialists in IT infrastructure
- > Certified, experienced, passionate

Based In Switzerland

- > 100% self-financed Swiss company
- > Over CHF10 mio. Turnover

Leading In Infrastructure Services

- > More than 150 customers in CH, D & F
- > Over 50 SLAs dbi FlexService contracted

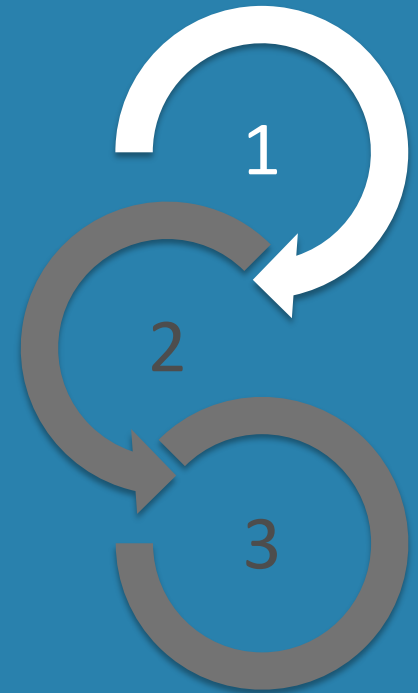
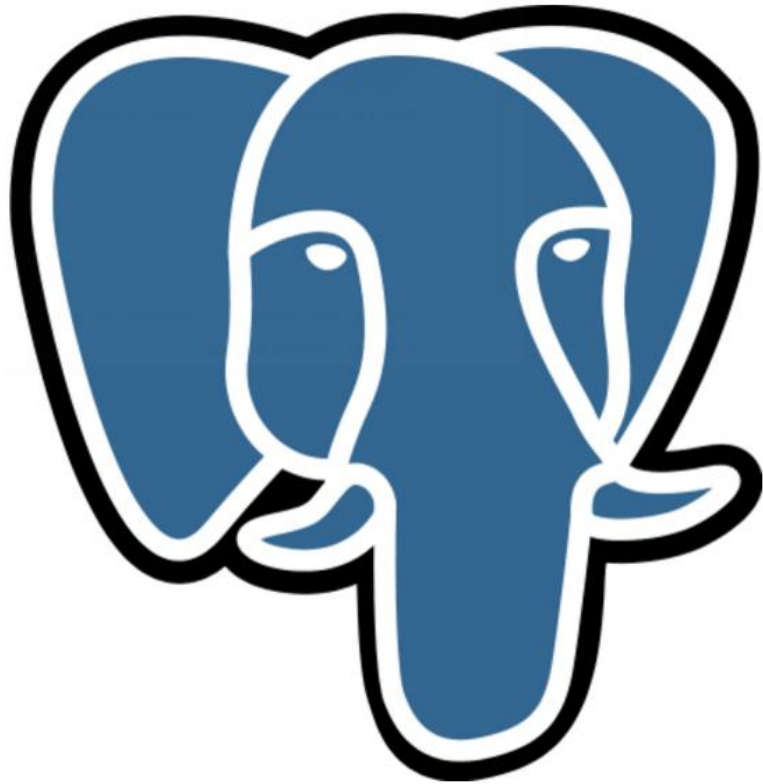


Agenda

1. How you can test development features
2. What is (probably) coming with PostgreSQL 11?
3. Conclusion



How you can test development features



How you can test development features

Nightly development snapshots



When you want to test something in the development tree

- > <https://www.postgresql.org/download/>
- > <https://www.postgresql.org/download/snapshots/>

BETA/RC RELEASES AND DEVELOPMENT SNAPSHOTS (UNSTABLE)

There are source code and binary **packages** of beta and release candidates and of the current development code available for testing and evaluation of new features. Note that these builds should be used **for testing purposes only**, and not for production systems.

How you can test development features

Nightly development snapshots

<https://www.postgresql.org/ftp/snapshot/dev>

DIRECTORIES

 [\[Parent Directory\]](#)

FILES

 postgresql-snapshot.tar.gz	2018-05-30 20:30:14	26.4 MB
 postgresql-snapshot.tar.bz2	2018-05-30 20:30:37	20.0 MB
 postgresql-snapshot.tar.bz2.md5	2018-05-30 20:35:45	62 bytes
 postgresql-snapshot.tar.bz2.sha256	2018-05-30 20:35:45	94 bytes
 postgresql-snapshot.tar.gz.md5	2018-05-30 20:35:45	61 bytes
 postgresql-snapshot.tar.gz.sha256	2018-05-30 20:35:46	93 bytes

How you can test development features

Nightly development snapshots

How can you run that?

```
$ wget https://ftp.postgresql.org/pub/snapshot/dev/postgresql-snapshot.tar.bz2
$ tar -axf postgresql-snapshot.tar.bz2
$ cd postgresql-*
$ ./configure --prefix=/tmp/postgresql-test
$ make -j 4 all
$ make install
$ cd contrib
$ make -j 4 install
$ /tmp/postgresql-test/bin/initdb -D /tmp/testpg
$ /tmp/postgresql-test/bin/pg_ctl -D /tmp/testpg -l /dev/null start
$ /tmp/postgresql-test/bin/psql -c "select version()" postgres

                version
-----
PostgreSQL 11beta1 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 , 64-bit
(1 row)
```

How you can test development features

Working with git

The PostgreSQL source code is managed in a git repository

> <https://git.postgresql.org/gitweb/>

> To clone the git repository

```
$ git clone git://git.postgresql.org/git/postgresql.git
$ cd postgresql
$ ./configure --prefix=/tmp/postgresql-test
$ make -j 4 all
$ make install
$ cd contrib
$ make -j 4 install
$ /tmp/postgresql-test/bin/initdb -D /tmp/testpg
$ /tmp/postgresql-test/bin/pg_ctl -D /tmp/testpg -l /dev/null start
$ /tmp/postgresql-test/bin/psql -c "select version()" postgres

                version
-----
PostgreSQL 11beta1 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 , 64-bit
(1 row)
```

How you can test development features

What to test



And now? What do you want to test?

```
$ /tmp/postgresql-test/bin/psql -X postgres
psql (11beta1)
Type "help" for help.

postgres=# select '???????';
 ?column?
-----
 ???????
(1 row)
```

How you can test development features

The development documentation

Either have a look at the development documentation

> <https://www.postgresql.org/docs/devel/static/index.html>

PostgreSQL 11beta1 Documentation

The PostgreSQL Global Development Group

Copyright © 1996-2018 The PostgreSQL Global Development Group

[Legal Notice](#)

Table of Contents

[Preface](#)

- [1. What is PostgreSQL?](#)
- [2. A Brief History of PostgreSQL](#)
- [3. Conventions](#)
- [4. Further Information](#)
- [5. Bug Reporting Guidelines](#)

How you can test development features

Git web interface

Or check the latest commits in git

> <https://git.postgresql.org/gitweb/?p=postgresql.git;a=summary>

shortlog				
17 hours ago	Alvaro Herrera	Fix grammar	master	commit commitdiff tree snapshot
19 hours ago	Teodor Sigaev	Move _bt_upgrademetapage() into critical section.	github/master	commit commitdiff tree snapshot
2 days ago	Peter Eisentraut	Initialize new jsonb iterator to zero		commit commitdiff tree snapshot
2 days ago	Andrew Dunstan	Return a value from Install.pm's lcopy function		commit commitdiff tree snapshot
2 days ago	Bruce Momjian	doc: mark 'replaceable' parameter for backup program...		commit commitdiff tree snapshot
2 days ago	Bruce Momjian	doc: adjust DECLARE docs to mention FOR UPDATE behavior		commit commitdiff tree snapshot
3 days ago	Andrew Dunstan	Avoid use of unportable hex constant in convutils.pm		commit commitdiff tree snapshot
3 days ago	Andrew Dunstan	Don't fall off the end of perl functions		commit commitdiff tree snapshot
3 days ago	Andrew Dunstan	Don't force a blank line before comments in perl code		commit commitdiff tree snapshot
4 days ago	Tom Lane	Update a couple of long-obsolete comments in pg_type.h.		commit commitdiff tree snapshot
5 days ago	Tom Lane	Fix misidentification of SQL statement type in plpgsql...		commit commitdiff tree snapshot
6 days ago	Magnus Hagander	Remove incorrect statement about IPC configuration...		commit commitdiff tree snapshot
6 days ago	Tom Lane	Update non-default collation tests for getObjectDescrip...		commit commitdiff tree snapshot
6 days ago	Tom Lane	Update seppgsql regression test output for getObjectDesc...		commit commitdiff tree snapshot
6 days ago	Tom Lane	Improve English wording of some other getObjectDescript...		commit commitdiff tree snapshot
6 days ago	Tom Lane	Improve translatability of some getObjectDescription...		commit commitdiff tree snapshot

How you can test development features

Git web interface

Select one which seems interesting

2018-05-15	Bruce Momjian	doc: clarify SCRAM channel binding	commit commitdiff tree snapshot
2018-05-14	Bruce Momjian	doc: update PG 11 release notes with suggested changes	commit commitdiff tree snapshot
2018-05-14	Alvaro Herrera	Don't allow partitioned index on foreign-table partitions	commit commitdiff tree snapshot
2018-05-14	Magnus Hagander	Fix file paths in comments	commit commitdiff tree snapshot
2018-05-14	Alvaro Herrera	psql: have \d show FKs on partitioned tables	commit commitdiff tree snapshot

Read the commit message and test

Don't allow partitioned index on foreign-table partitions

Creating indexes on foreign tables is already forbidden, but local partitioned indexes (commit [8b08f7d4820f](#)) forgot to check for them. Add a preliminary check to prevent wasting time.

Another school of thought says to allow the index to be created if it's not a unique index; but it's possible to do better in the future (enable indexing of foreign tables, somehow), so we avoid painting ourselves in a corner by rejecting all cases, to avoid future grief (a.k.a. backward incompatible changes).

Reported-by: Arseny Sher

Author: Amit Langote, Álvaro Herrera

Discussion: <https://postgr.es/m/87sh71cakz.fsf@ars-thinkpad>

How you can test development features

Testing features not yet committed



But what when you want to test something that is not yet committed?

> <https://commitfest.postgresql.org/>

Commitfests

The following commitfests exist in the system. New patches should be submitted to commitfest **2018-09**.

- **2018-11** (Future - 2018-11-01 - 2018-11-30)
- **2018-09** (Open - 2018-09-01 - 2018-09-30)



How you can test development features

Testing features not yet committed

Select one which seems interesting

Monitoring & Control		
pg_stat_statements with plans (v02)	Needs review	Marius Timmer (mtimm_01), Julian Markwort (markwort)

Read, apply patch and test

Emails

[\[FEATURE PATCH\] pg_stat_statements with plans](#) ✕

First at [2017-01-25 17:34:49](#) by Julian Markwort <julian.markwort at uni-muenster.de>

Latest at [2018-03-25 21:17:01](#) by Arthur Zakirov <a.zakirov at postgrespro.ru>

Latest attachment [\(pgss_plans_v05.patch\)](#) at [2018-03-20 21:44:21](#) from Julian Markwort <julian.markwort at uni-muenster.de> +

[Re: \[FEATURE PATCH\] pg_stat_statements with plans \(v02\)](#) ✕

First at [2018-04-06 13:03:30](#) by Julian Markwort <julian.markwort at uni-muenster.de>

Latest at [2018-04-25 18:23:42](#) by Robert Haas <robertmhaas at gmail.com>

Latest attachment [\(pgss_plans_v06.patch\)](#) at [2018-04-06 13:03:30](#) from Julian Markwort <julian.markwort at uni-muenster.de> +

How you can test development features

Testing features not yet committed



How to apply a patch?

> https://wiki.postgresql.org/wiki/Working_with_Git

```
$ git checkout -b feature-to-review  
$ patch -p1 < feature.patch
```

How you can test development features

Testing features not yet committed

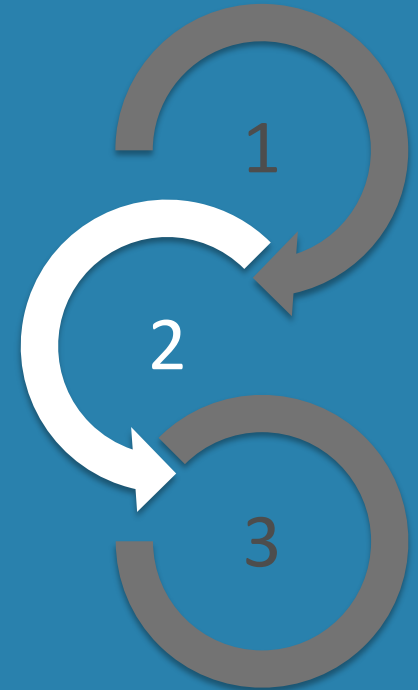
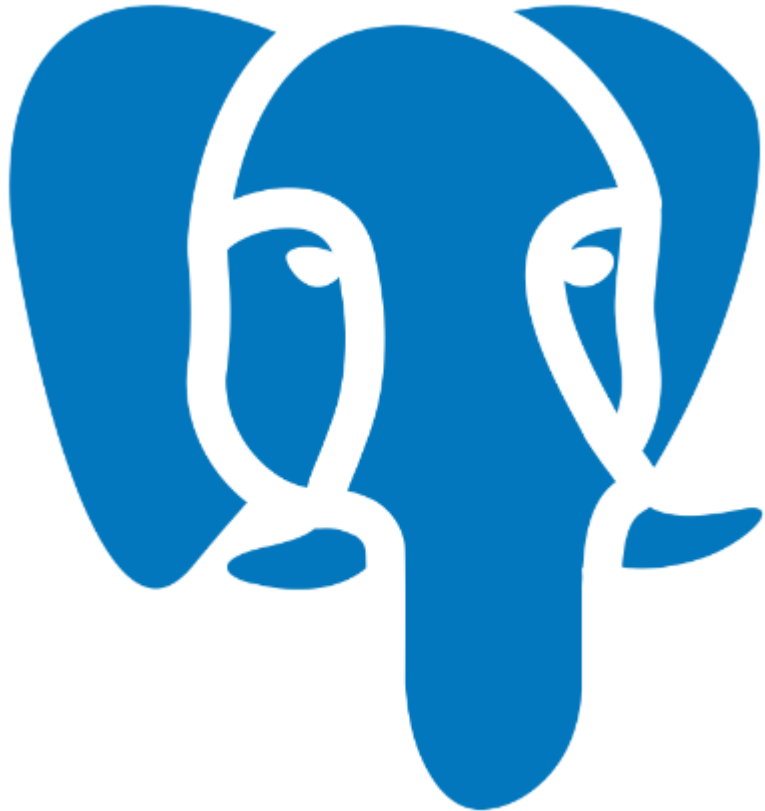
Next year, probably end of June, probably exactly here ...

It is up to someone of you to present the PostgreSQL 12 new features!



No, no excuses, now you know how to do it!

What is (probably) coming with PostgreSQL11



Demo time

See it, feel it, understand it

We will skip this chapter and do a demo instead

> What is in the demo is also on the slides



Did you ever face this?

```
$ psql -X postgres
psql (10.3)
Type "help" for help.

postgres=# addad
postgres-# adass
postgres-# exit
postgres-# quit
postgres-# let me out
```

What most newcomers miss

```
$ psql -X postgres
psql (10.3)
Type "help" for help.
```

What most newcomers miss

```
$ psql -X postgres
psql (10.3)
Type "help" for help.

postgres=# help
You are using psql, the command-line interface to PostgreSQL.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit
```

My favorite new feature for those not willing to read

```
$ psql -X postgres
psql (11devel)
Type "help" for help.

postgres=# exit

$ psql -X postgres
psql (11devel)
Type "help" for help.

postgres=# quit

$
```

My favorite new feature for those not willing to read

```
$ psql -X postgres
psql (11devel)
Type "help" for help.

postgres=# assa
postgres-# asas
postgres-# quit
Use \q to quit.
postgres-#
```

What happens when you try to update the partition key in PostgreSQL 10?

```
$ psql -X postgres
psql (10.0)
Type "help" for help.

postgres=# select version();
postgres=# create table part ( a int, list varchar(5) ) partition by list (list);
postgres=# create table part_1 partition of part for values in ('beer');
postgres=# create table part_2 partition of part for values in ('wine');
postgres=# \d+ part
postgres=# insert into part (a,list) select i,'beer' from generate_series(1,10) i;
postgres=# insert into part (a,list) select i,'wine' from generate_series(11,20) i;
postgres=# update part set list = 'beer' where a = 15;

ERROR:  new row for relation "part_2" violates partition constraint
DETAIL:  Failing row contains (15, beer).
```

PostgreSQL 11 new features

Partitioning improvements

Updating the partition key now works in PostgreSQL 11

```
$ psql -X postgres
psql (11devel)
Type "help" for help.

postgres=# select version();
postgres=# create table part ( a int, list varchar(5) ) partition by list (list);
postgres=# create table part_1 partition of part for values in ('beer');
postgres=# create table part_2 partition of part for values in ('wine');
postgres=# \d+ part
postgres=# insert into part (a,list) select i,'beer' from generate_series(1,10) i;
postgres=# insert into part (a,list) select i,'wine' from generate_series(11,20) i;
postgres=# update part set list = 'beer' where a = 15;
UPDATE 1
```

What happens when you try to create an index on a partitioned table in PostgreSQL 10?

```
$ psql -X postgres
psql (11devel)
Type "help" for help.

postgres=# select version();
postgres=# create table part ( a int, list varchar(5) ) partition by list (list);
postgres=# create table part_1 partition of part for values in ('beer');
postgres=# create table part_2 partition of part for values in ('wine');
postgres=# \d+ part
postgres=# insert into part (a,list) select i,'beer' from generate_series(1,10) i;
postgres=# insert into part (a,list) select i,'wine' from generate_series(11,20) i;
postgres=# create index i_test on part (a);
ERROR:  cannot create index on partitioned table "part"
```

PostgreSQL 11 new features

Partitioning improvements

Creating an index on a partitioned table works PostgreSQL 11

```
$ psql -X postgres
psql (11devel)
Type "help" for help.

postgres=# select version();
postgres=# create table part ( a int, list varchar(5) ) partition by list (list);
postgres=# create table part_1 partition of part for values in ('beer');
postgres=# create table part_2 partition of part for values in ('wine');
postgres=# \d+ part
postgres=# insert into part (a,list) select i,'beer' from generate_series(1,10) i;
postgres=# insert into part (a,list) select i,'wine' from generate_series(11,20) i;
postgres=# create index i_test on part (a);
CREATE INDEX
```

PostgreSQL 11 new features

Partitioning improvements

Insert on conflict...do, against a partitioned table in PostgreSQL 10

```
$ psql -X postgres
psql (11devel)
Type "help" for help.

postgres=# select version();
postgres=# create table part ( a int, list varchar(5) ) partition by list (list);
postgres=# create table part_1 partition of part for values in ('beer');
postgres=# create table part_2 partition of part for values in ('wine');
postgres=# \d+ part
postgres=# insert into part (a,list) select i,'beer' from generate_series(1,10) i;
postgres=# insert into part (a,list) select i,'wine' from generate_series(11,20) i;
postgres=# insert into part(a,list) values (1,'beer')
           on conflict (a,list) do nothing;
```

ERROR: ON CONFLICT clause is not supported with partitioned tables

PostgreSQL 11 new features

Partitioning improvements

Insert on conflict...do, in PostgreSQL 11

```
$ psql -X postgres
psql (11devel)
Type "help" for help.

postgres=# select version();
postgres=# create table part ( a int, list varchar(5) ) partition by list (list);
postgres=# create table part_1 partition of part for values in ('beer');
postgres=# create table part_2 partition of part for values in ('wine');
postgres=# \d+ part
postgres=# insert into part (a,list) select i,'beer' from generate_series(1,10) i;
postgres=# insert into part (a,list) select i,'wine' from generate_series(11,20) i;
postgres=# insert into part(a,list) values (1,'beer')
           on conflict (a,list) do nothing;

INSERT 0 0
```

What is partition pruning?

Avoiding to scan partitions which are known to not contain any matching tuples

- > In PostgreSQL 10 this can be only done at planning time
- > PostgreSQL 11 will be able to do partition pruning at execution time as well

Support partition pruning at execution time

```
author      Alvaro Herrera <alvherre@alvh.no-ip.org>
            Sat, 7 Apr 2018 22:54:31 +0200 (17:54 -0300)
committer   Alvaro Herrera <alvherre@alvh.no-ip.org>
            Sat, 7 Apr 2018 22:54:39 +0200 (17:54 -0300)
commit      499be013de65242235ebdde06adb08db887f0ea5
tree        c1f69b818f917379fb4f72e80535fef899a40b5b   tree | snapshot
parent      5c0675215e153ba1297fd494b34af2fdebd645d1   commit | diff
```

PostgreSQL 11 new features

Partitioning improvements

PostgreSQL 10 can already do that at planning time

```
$ psql -X postgres
psql (11devel)
Type "help" for help.

postgres=# select version();
postgres=# create table part ( a int, list varchar(5) ) partition by list (list);
postgres=# create table part_1 partition of part for values in ('beer');
postgres=# create table part_2 partition of part for values in ('wine');
postgres=# \d+ part
postgres=# insert into part (a,list) select i,'beer' from generate_series(1,10) i;
postgres=# insert into part (a,list) select i,'wine' from generate_series(11,20) i;
postgres=# explain select * from part where list = 'beer';
```

QUERY PLAN

```
-----
Append  (cost=0.00..28.12 rows=7 width=28)
  -> Seq Scan on part_1  (cost=0.00..28.12 rows=7 width=28)
    Filter: ((list)::text = 'beer'::text)
```

PostgreSQL 11 new features

Partitioning improvements

PostgreSQL 10 can not do this (subquery)

```
postgres=# explain select * from part where list = ( select 'beer' );  
               QUERY PLAN
```

```
-----  
Append  (cost=0.01..56.26 rows=14 width=28)  
  InitPlan 1 (returns $0)  
    -> Result  (cost=0.00..0.01 rows=1 width=32)  
    -> Seq Scan on part_1  (cost=0.00..28.12 rows=7 width=28)  
        Filter: ((list)::text = $0)  
    -> Seq Scan on part_2  (cost=0.00..28.12 rows=7 width=28)  
        Filter: ((list)::text = $0)
```

Doing the same in PostgreSQL 11

```
postgres=# explain select * from part where list = ( select 'beer' );
               QUERY PLAN
-----
 Append  (cost=0.01..56.26 rows=14 width=28)
   InitPlan 1 (returns $0)
     -> Result  (cost=0.00..0.01 rows=1 width=32)
   -> Seq Scan on part_1  (cost=0.00..28.12 rows=7 width=28)
       Filter: ((list)::text = $0)
   -> Seq Scan on part_2  (cost=0.00..28.12 rows=7 width=28)
       Filter: ((list)::text = $0)
```

Didn't I tell you that works in PostgreSQL 11

Pruning at execution times means the query must be executed

```
postgres=# explain (analyze) select * from part where list = ( select 'beer' );
```

QUERY PLAN

```
-----
Append  (cost=0.01..56.33 rows=14 width=28) (actual time=0.010..0.010 rows=0 loops=1)
  InitPlan 1 (returns $0)
    -> Result  ( rows=1 width=32) (actual time=0.001..0.001 rows=1 loops=1)
    -> Seq Scan on part_1  (rows=7 width=28) (actual time=0.003..0.003 rows=0 loops=1)
        Filter: ((list)::text = $0)
    -> Seq Scan on part_2  (cost=0.00..28.12 rows=7 width=28) (never executed)
        Filter: ((list)::text = $0)
```

```
postgres=# show enable_partition_pruning;
enable_partition_pruning
```

```
-----
on
```

PostgreSQL 11 will come with hash partitioning

```
postgres=# create table part2 ( a int, list varchar(5) ) partition by hash (a);
postgres=# create table part2_1
           partition of part2 FOR VALUES WITH (MODULUS 3, REMAINDER 0);
postgres=# create table part2_2
           partition of part2 FOR VALUES WITH (MODULUS 3, REMAINDER 1);
postgres=# create table part2_3
           partition of part2 FOR VALUES WITH (MODULUS 3, REMAINDER 2);
postgres=# insert into part2 (a,list) values (1,'beer');
postgres=# insert into part2 (a,list) values (2,'whine');
postgres=# insert into part2 (a,list) values (3,'schnaps');
```

> Tuples will be evenly distributed by the remainder of the modules

PostgreSQL 11 new features

Partitioning improvements

PostgreSQL 11 will come with hash partitioning, **evenly distributed data**

```
postgres=# select * from only part2_1;
```

```
 a | list  
---+-----  
  2 | whine  
(1 row)
```

```
postgres=# select * from only part2_2;
```

```
 a | list  
---+-----  
  3 | schnaps  
(1 row)
```

```
postgres=# select * from only part2_3;
```

```
 a | list  
---+-----  
  1 | beer  
(1 row)
```

PostgreSQL 11 will have support for indexes with include columns

```
postgres=# create table tt (a int, b text, c date );
postgres=# insert into tt (a,b,c)
postgres=# select aa.*, md5(aa::text), now()
postgres=# from generate_series(1,1000000) aa;
postgres=# create unique index i1 on tt(a) include(b);
postgres=# \d+ tt
```

Table "public.tt"

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
a	integer				plain		
b	text				extended		
c	date				plain		

Indexes:

```
"i1" UNIQUE, btree (a) INCLUDE (b)
```

PostgreSQL 11 new features

New hashing functions

New SHA hashing functions show up in PostgreSQL 11

```
$ psql -X postgres
psql (11devel)
Type "help" for help.
```

```
postgres=# \df sha*
```

List of functions

Schema	Name	Result data type	Argument data types	Type
pg_catalog	sha224	bytea	bytea	func
pg_catalog	sha256	bytea	bytea	func
pg_catalog	sha384	bytea	bytea	func
pg_catalog	sha512	bytea	bytea	func

(4 rows)

PostgreSQL 11 new features

Procedures with transaction control



In PostgreSQL 10 you can create functions which return void/nothing

```
postgres=# CREATE FUNCTION dummy_func (id int) RETURNS VOID AS $$  
DECLARE  
BEGIN  
RAISE NOTICE 'id is %', id;  
END;  
$$ LANGUAGE plpgsql;
```

... but you can not do this:

```
postgres=# CREATE PROCEDURE dummy_proc (id int) AS $$  
DECLARE  
BEGIN  
    raise notice 'id is %', id;  
END;  
$$ LANGUAGE plpgsql;
```

PostgreSQL 11 new features

Procedures with transaction control

PostgreSQL 11 comes with support for procedures

```
postgres=# CREATE PROCEDURE dummy_proc (id int) AS $$  
DECLARE  
BEGIN  
    raise notice 'id is %', id;  
END;  
$$ LANGUAGE plpgsql;  
  
postgres=# call dummy_proc(1);  
NOTICE:  id is 1  
CALL
```

PostgreSQL 11 new features

Procedures with transaction control

PostgreSQL 11 comes with support for procedures & transaction control

```
postgres=# create table t1 ( a int primary key );
postgres=# CREATE OR REPLACE PROCEDURE dummy_proc2 (id int) AS $$
DECLARE
l_id int := id;
BEGIN
    raise notice 'id is %', l_id;
    insert into t1 (a) values (l_id);
    commit;
    l_id := l_id + 1;
    raise notice 'id is %', l_id;
    insert into t1 (a) values (l_id);
    commit;
    raise notice 'id is %', l_id;
    insert into t1 (a) values (l_id);
END;
$$ LANGUAGE plpgsql;
```

PostgreSQL 11 comes with support for procedures & transaction control

```
postgres=# call dummy_proc2(1);
NOTICE: id is 1
NOTICE: id is 2
NOTICE: id is 2
ERROR: duplicate key value violates unique constraint "t1_pkey"
DETAIL: Key (a)=(2) already exists.
CONTEXT: SQL statement "insert into t1 (a) values (l_id)"
PL/pgSQL function dummy_proc2(integer) line 13 at SQL statement

postgres=# select * from t1;
 a
---
 1
 2
(2 rows)
```

PostgreSQL 11 new features

Faster add column with a non null default value



Up to PostgreSQL 10, when you did this:

```
postgres=# create table ttt (a int, b text, c date );
postgres=# insert into ttt (a,b,c)
           select aa.*, md5(aa::text), now()
           from generate_series(1,1000000) aa;
postgres=# \timing
Timing is on.
postgres=# alter table ttt add column d text default 'a' ;
ALTER TABLE
Time: 1307.764 ms (00:01.308)
```

> The entire table gets re-written

PostgreSQL 11 new features

Faster add column with a non null default value



In PostgreSQL 11 adding a column with a default value is instant

```
postgres=# create table ttt (a int, b text, c date );
postgres=# insert into ttt (a,b,c)
           select aa.*, md5(aa::text), now()
           from generate_series(1,1000000) aa;
postgres=# \timing
Timing is on.
postgres=# alter table ttt add column d text default 'a' ;
ALTER TABLE
Time: 20.949 ms
```

> But how can that work?

PostgreSQL 11 new features

Faster add column with a non null default value

In PostgreSQL 11 pg_attribute got two new columns

```
postgres=# \d pg_attribute
Table "pg_catalog.pg_attribute"
  Column          |  Type   | Collation | Nullable | Default
-----+-----+-----+-----+-----
 attrelid         | oid     |           | not null |
 attname         | name    |           | not null |
 atttypid        | oid     |           | not null |
 attstattarget    | integer |           | not null |
 ...
 atthasmissing   | boolean |           | not null |
 ...
 attinhcount      | integer |           | not null |
 attcollation     | oid     |           | not null |
 attacl          | aclitem[] |           |          |
 attoptions       | text[]  |           |          |
 attfdwoptions    | text[]  |           |          |
 attmissingval  | anyarray |           |          |
```

PostgreSQL 11 new features

Faster add column with a non null default value

Once a column is added with a default value those columns get set

```
postgres=# select attmissingval
           , atthasmissing
           from pg_attribute
           where attrelid = 'ttt'::regclass
           and attname = 'd';
 attmissingval | atthasmissing 
-----+-----
      {a}      | t
(1 row)
```

- > Every row that is updated and older than the change of the table will get the new value set
- > Every row that is selected and older than the change of the table will get the new value set
- > New rows get it set anyway
- > Once the table is re-written the flags are cleared

PostgreSQL 11 new features

JIT compilation



PostgreSQL will come with JIT compilation

- > JIT = Just-in-time
- > Turning interpreted programs into machine code at runtime

Example

- > Instead of having "something" than can evaluate WHERE a = 3
- > Create a function that natively can be executed on the CPU to just do that

PostgreSQL needs to be build with "--with-lvm"

PostgreSQL 11 new features

JIT compilation

There are a few parameters which control JIT

```
postgres=# select name
           , setting
           from pg_settings
           where name like 'jit%';
```

name	setting
jit	on
jit_above_cost	100000
jit_debugging_support	off
jit_dump_bitcode	off
jit_expressions	on
jit_inline_above_cost	500000
jit_optimize_above_cost	500000
jit_profiling_support	off
jit_provider	llvmjit
jit_tuple_deforming	on

(10 rows)

A (fake) JIT example

```
postgres=# set jit_above_cost=5;  
postgres=# set jit_optimize_above_cost=5;  
postgres=# set jit_inline_above_cost=5;  
postgres=# explain select sum(a) from ttt;
```

QUERY PLAN

```
-----  
Finalize Aggregate  (cost=15554.55..15554.56 rows=1 width=8)  
  -> Gather  (cost=15554.33..15554.54 rows=2 width=8)  
        Workers Planned: 2  
          -> Partial Aggregate  (cost=14554.33..14554.34 rows=1 width=8)  
                -> Parallel Seq Scan on ttt  (cost=0.00..13512.67 rows=416667 width=4)
```

JIT:

```
Functions: 8  
Inlining: true  
Optimization: true
```

(9 rows)

New psql variables for ERROR, SQLSTATE and ROWCOUNT

```
postgres=# select count(*) from ttt;
 count
-----
1000000

postgres=# \echo :ROW_COUNT
1
postgres=# \echo :SQLSTATE
00000
postgres=# select count(*) from ttt;
ERROR:  syntax error at or near "."
LINE 1: select count(*) from ttt;
                  ^
postgres=# \echo :ERROR
True
postgres=# \echo :SQLSTATE
postgres=# \! grep 42601 $PGHOME/share/errcodes.txt
postgres=# \! vi $PGHOME/share/errcodes.txt
```

PostgreSQL 11 new features

Parallel (concurrent) btree index build

PostgreSQL 11 supports parallel (concurrent) btree index builds

> The parameters which control that

```
postgres=# show max_parallel_workers;
max_parallel_workers
-----
8

postgres=# show max_parallel_maintenance_workers;
max_parallel_maintenance_workers
-----
2

postgres=# show maintenance_work_mem;
maintenance_work_mem
-----
64MB
```

PostgreSQL 11 new features

Parallel (concurrent) btree index build



You will not see it

```
postgres=# create index ip on iptest (a,b,c);  
CREATE INDEX  
postgres=#
```

- > Was it parallel or serial?
- > Of course you can disable the parallel operations and compare time

PostgreSQL 11 new features

Parallel (concurrent) btree index build

Or you open a second session before the index build

```
postgres=# select query,state,backend_type from pg_stat_activity;
postgres=# \watch
```

Create the index

```
postgres=# create index ip on iptest (a,b,c);
CREATE INDEX
```

Check the first session

```
postgres=# select query,state,backend_type from pg_stat_activity;
               query               | state | backend_type
-----+-----+-----
...
create index ip on iptest (a,b,c); | active | client backend
select query,state,backend_type from pg_stat_activity; | active | client backend
create index ip on iptest (a,b,c); | active | parallel worker
...
```

With PostgreSQL 10 you can get a plan like this:

```
postgres=# explain select max(t1.a) from iptest t1, iptest t2
           where t1.a = t2.a;
               QUERY PLAN
-----
Finalize Aggregate  (cost=1036388347.67..1036388347.68 rows=1 width=32)
  -> Gather  (cost=1036388347.45..1036388347.66 rows=2 width=32)
        Workers Planned: 2
        -> Partial Aggregate  (cost=1036387347.45..1036387347.46 rows=1 width=32)
              -> Merge Join  (cost=1429874.39..888539787.42 rows=59139024013 width=32)
                    Merge Cond: (t1.a = t2.a)
                    -> Sort  (cost=444407.57..449957.49 rows=2219967 width=32)
                          Sort Key: t1.a
                          -> Parallel Seq Scan on iptest t1
                    -> Materialize  (cost=985466.82..1012106.42 rows=5327920 width=32)
                          -> Sort  (cost=985466.82..998786.62 rows=5327920 width=32)
                                Sort Key: t2.a
                                -> Seq Scan on iptest t2
```

PostgreSQL 11 comes with parallel hash joins

```
postgres=# explain select max(t1.a) from iptest t1, iptest t2
           where t1.a = t2.a;
```

QUERY PLAN

```
-----
Finalize Aggregate  (cost=322058.98..322058.99 rows=1 width=32)
  -> Gather  (cost=322058.77..322058.98 rows=2 width=32)
        Workers Planned: 2
        -> Partial Aggregate  (cost=321058.77..321058.78 rows=1 width=32)
              -> Parallel Hash Join  (cost=145120.45..315850.38 rows=2083353 width=33)
                    Hash Cond: (t1.a = t2.a)
                    -> Parallel Seq Scan on iptest t1
                    -> Parallel Hash  (cost=102801.53..102801.53 rows=2083353 width=33)
                          -> Parallel Seq Scan on iptest t2
```

PostgreSQL 11 new features

WALSIZE with initdb



In PostgreSQL 10 you can specify the WALSEGSize only at compile time

```
postgres@pgbox:~# ./configure --prefix=${PGHOME} \
                        --with-wal-segsize=${WALSEGSize}
```

- > Once it was compiled and installed this is fixed
- > Changes to the WAL segment size require new binaries to be compiled

In PostgreSQL 11 this becomes configurable with initdb

```
postgres@pgbox:~# initdb --help
initdb initializes a PostgreSQL database cluster.

Usage:
  initdb [OPTION]... [DATADIR]

Options:
...
  -W, --pwprompt          prompt for a password for the new superuser
  -X, --waldir=WALDIR     location for the write-ahead log directory
      --wal-segsize=SIZE  size of WAL segments, in megabytes
```

> The same binaries can be used for clusters with different WAL segment size configurations

PostgreSQL 11 new features

Multiple table arguments for vacuum and analyze

In PostgreSQL 10 you can only vacuum/analyze one table at a time

```
postgres=# vacuum part;  
VACUUM  
postgres=# vacuum ttt;  
VACUUM  
postgres=# vacuum ttt,part;  
ERROR:  syntax error at or near ", "  
LINE 1: vacuum ttt,part;
```

PostgreSQL 11 new features

Multiple table arguments for vacuum and analyze

PostgreSQL 11 allows to do that on multiple tables

```
postgres=# vacuum part;  
VACUUM  
postgres=# vacuum ttt;  
VACUUM  
postgres=# vacuum ttt,part;  
VACUUM  
postgres=# analyze ttt,part;  
ANALYZE
```

PostgreSQL 11 new features

Specify a separate PAGER for psql



A new environment variable PSQL_PAGER controls the PAGER psql will use

```
postgres@pgbox:~# echo $PAGER
sed "s/^\\([0-9]\\+ [rows]\\+\\)/\\1/;s/^\\(-\\[\\ RECORD\\ [0-9]\\+\\ \\)[-\\+\\]\\+\\)/\\1/;s/|/|/g;s/^\\([-+]\\+\\)/\\1/" 2>/dev/null | less
postgres@pgbox:~# psql -c "select * from pg_stat_attribute" postgres
postgres@pgbox:~# export PSQL_PAGER=cat
postgres@pgbox:~# psql -c "select * from pg_stat_attribute" postgres
```

> PAGER is still used when PSQL_PAGER is not set

PostgreSQL 11 new features

Restoring shared buffer content automatically after restart

The `pg_prewarm` extension in PostgreSQL 11 can restore the shared buffer content automatically

```
postgres=# create extension pg_prewarm;
postgres=# create extension pg_buffercache;
postgres=# alter system set shared_preload_libraries='pg_prewarm';
postgres=# \! pg_ctl -D $PGDATA restart -m fast
postgres=# \! ps -ef | grep warm
postgres=# select * from ttt limit 30;
postgres=# SELECT pg_relation_filepath('ttt');
postgres=# select count(*) from pg_buffercache where relfilenode = [FILE_NODE];
postgres=# \! pg_ctl -D $PGDATA restart -m fast
postgres=# select count(*) from pg_buffercache where relfilenode = [FILE_NODE];
postgres=# select 'database,tablespace,table,fork,block';
postgres=# \! cat $PGDATA/autoprewarm.blocks | more
```

PostgreSQL 11 new features

Allow locking of views by locking an underlying table

Before PostgreSQL 11 locking an underlying table does not lock the view on top of it

```
postgres=# create table tt1 ( a int );
CREATE TABLE
postgres=# create view vv1 as select * from tt1;
CREATE VIEW
postgres=# begin;
BEGIN
postgres=# lock table tt1;
LOCK TABLE
```

```
postgres=# select count(*) from vv1;
 count
-----
      0
```

PostgreSQL 11 new features

Allow locking of views by locking an underlying table

PostgreSQL 11 by default locks the view

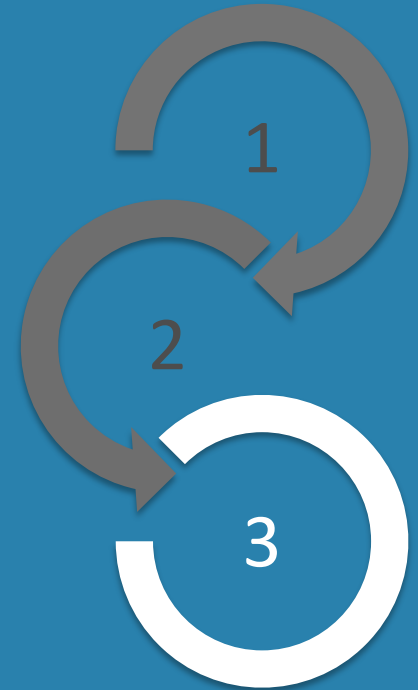
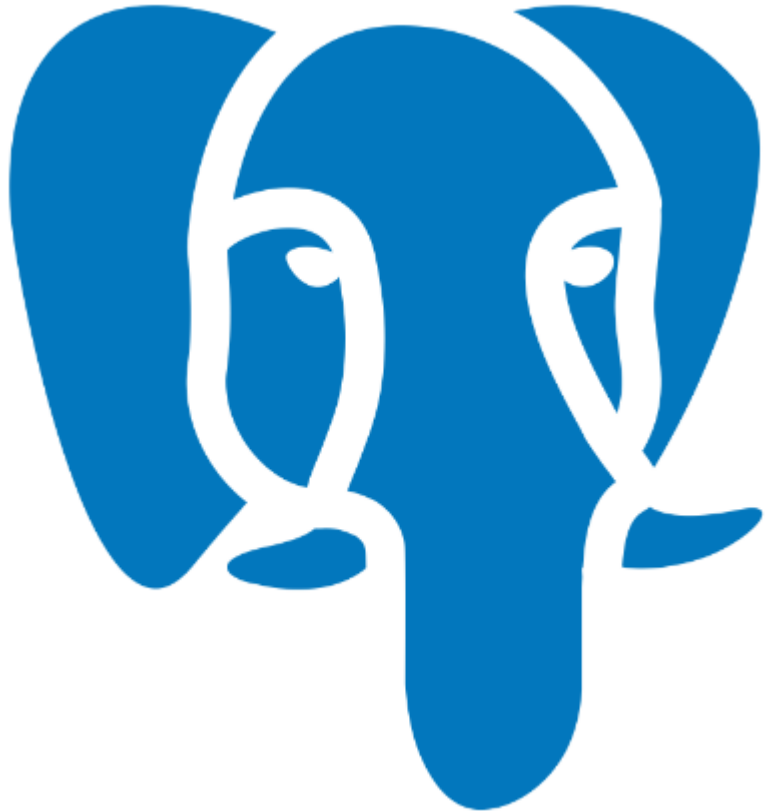
```
postgres=# create table tt1 ( a int );
CREATE TABLE
postgres=# create view vv1 as select * from tt1;
CREATE VIEW
postgres=# begin;
BEGIN
postgres=# lock table tt1;
LOCK TABLE
```

1

```
postgres=# select count(*) from vv1;
-- BLOCKS
```

2

Conclusion



As always, many, many improvements in each release

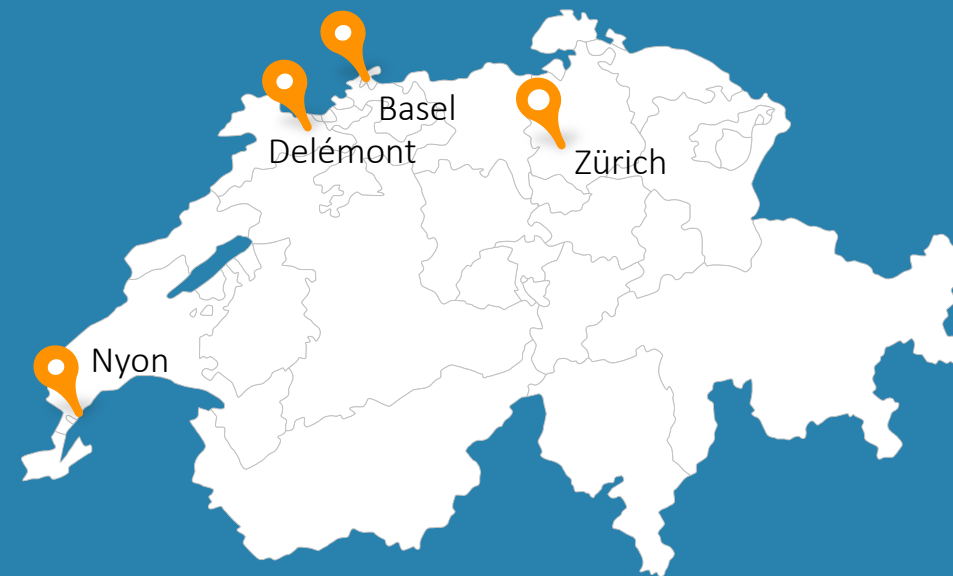
Full list of changes here:

> <https://www.postgresql.org/docs/11/static/release-11.html#id-1.11.6.5.5>

You must read the release notes, **always!!!**

Any questions?

Please do ask!



We would love to boost
your IT-Infrastructure
How about you?